Question 1

1. Plan A: \( T(X_1) = 100,000 \) \( T(X_2) = 100,000 \) \( T(X_3) = 10,000 \)
   Plan B: \( T(X_1) = 20 \) \( T(X_2) = 10,000 \) \( T(X_3) = 10,000 \)

2. \( \text{Cost(Plan}_B\text{)} = \text{Cost}(S) + \text{Cost}(SH) + \text{Cost}(S, SH) + \text{Cost}(P) + \text{Cost}(S, SH, P) \)
   \( = (20 + 1) + 20(100,000 + 1) + (10,000 + 1) + 10,000(100 + 1) + (10,000 + 1) \)

3. Here, the most important thing to figure out was how much memory we should allocate
   for bucketizing. You must also note that because we are pipelining, the memory is
   shared throughout all the joins. For the given information, the optimal cost was when
   \( k=10 \). And the total cost would be:
   \( \text{Cost(Plan}_A\text{)} = \text{Read}(S) + \text{Read}(SH) + \text{WriteRemainingBuckets}(S, SH) + \text{Read}(P) + \text{WriteRemainingBuckets}(P) + \text{ReadRemainingBuckets}(S, SH) + \text{ReadRemainingBuckets}(P) \)
   \( = 1 + 174 + (388 - 39) + 20 + 18 + (388 - 39) + 18 = 929 \)

4. \( \text{Cost(Plan}_B\text{)} = 195 \)

5. The most optimal plan was a right deep plan of \( P \bowtie S \bowtie SH \)

Question 2
1. If the constraint is reduced, more sparse leaf nodes will become present. As a result, the number of internal nodes, and hence the tree height can become higher.

2. If we increase the bound, the complexity of any adjustments, e.g. after deleting, will be increased.

**Question 3**

One can argue that both sort-merge-moin and hash join can be extended easily for full outerjoins. However, ultimately, sort-merge-join has a slight advantage, as it will require less complex computation to find out whether natural join cannot be performed and to add the null element.

**Question 4**

As cartesian products are not considered, the only location of the S table in the join tree is in the two leftmost positions, i.e. it must be joined either as the first or second table. If S is joined as the leftmost table, there are n! possibilities to perturb the R tables. If S is joined as the second leftmost table, there are n possibilities to choose the first table, and the rest can be perturbed by (n-1)! possibilities. Hence n! + n(n-1)! = 2n!

**Question 5**

If you note three following characteristics about Selinger optimizer, this question is trivial. a) It prunes plans on each level b) Only left deep plans are considered and c) No cartesian joins are considered.

Hence, the answer would look like the following:

- **Level 1**: Segment scan(T), Index scan(T), Segment scan(S), Segment scan(R), where Segment scan(T) will be pruned
- **Level 2**: TNLJ(S,R), TNLJ(R,S), SMJ(S,R), TNLJ(T,R), TNLJ(R,T), SMJ(R,T).

As SMJ(sort-merge-join) is the lowest cost join for both $R \bowtie S$ and $R \bowtie T$, all the TNLJ’s(Tuple nested loop join) will be pruned

- **Level 3**: TNLJ(SMJ(S,R),T), SMJ(SMJ(S,R),T), TNLJ(SMJ(R,T),S), SMJ(SMJ(R,T),S)

**Question 6**

4096/8=512 is the number of elements(key, pointer) that can be contained in a block.

1. For an internal node with “a” keys, there must be “a” pointers to the data and “a+1” keys to the next level of the tree. Hence $a + a + (a + 1) \leq 512 \Rightarrow a \simeq 170$

2. For a leaf node with “a” keys, there must be “a+1” pointers, if we consider a pointer to the sibling. Hence $a + (a + 1) \leq 512 \Rightarrow a \simeq 255$. 


3. You can argue from both sides. On one hand, because you cannot do a range search in the $B_-$ tree like in normal B tree, as you must visit the parents in between, having a pointer to sibling wouldn’t help. However, on the other hand, you can accomplish a range search by doing a range search on each level instead of doing it only on the leaves. In that case a pointer to the sibling is useful.

4. 4 levels.